Database Project

Rishikesh Sahil 220892 rishikeshs22@iitk.ac.in
Kawaljeet Singh 220514 kawaljeets22@iitk.ac.in
Nikhil Jain 220709 nikhilj22@iitk.ac.in
Abhishek Kumar 220041 kumara22@iitk.ac.in

24th April 2025

1 Summary

This report presents a detailed design and analysis of a hospital management relational database system. The database is structured to effectively manage core hospital operations such as patient care, medical staff assignments, room allocation, billing, insurance processing, medical history tracking, prescriptions, and laboratory tests. The design emphasizes normalization (Third Normal Form), data integrity, consistency, and scalability while providing support for future integration with external health systems and analytics tools. We have made sure that it follows ACID properties.

2 Motivation and Problem Statement

Hospitals deal with massive amounts of structured and semi-structured data related to patients, staff, treatments, tests, and finances. Traditional paper-based or fragmented systems result in data duplication, errors, delays in patient care, and administrative inefficiencies.

The motivation for this project comes from the need to consolidate hospital data in a centralized relational system to:

- Reduce redundancy and inconsistency in patient and staff data.
- Make data more effective by splitting bigger databases into smaller so that insertion and deletion of all the hospital databases more cost effective.
- Enable real-time access to medical records, prescriptions, appointments, and billing information.
- Facilitate regulatory compliance (e.g., data audits, insurance claims) in a consistent manner.
- Support decision making by integrating clinical and operational data.

The primary problem statement for us is to create a well-integrated database schema that reflects real-world hospital workflows and enforces data integrity through sound relational design principles.

3 Methodology

The database design follows the Entity-Relationship (ER) model, later translated into a normalized relational schema. Each table represents a distinct entity or relationship in the hospital system. Key design decisions include the following.

4 Entities

The following subsections detail the 16 entities in the `HospitalDB` schema. Each entity is described, and its attributes are presented in a table format, including data types and constraints.

4.1 Patient

Description: Represents individuals receiving medical care at the hospital. Stores personal details such as name, contact information, blood type, gender, and medical condition.

Table 1: Patient Attributes

Attribute	**Data Type**	**Constraints**
Patient_ID	int	Primary Key
Patient_FName	varchar(20)	
Patient_LName	varchar(20)	
Phone	varchar(13)	
Blood_Type	varchar(5)	
Email	varchar(50)	
Gender	varchar(10)	
Condition	varchar(30)	

4.2 Department

Description: Represents organizational units within the hospital (e.g., Cardiology, Neurology). Stores department head and name.

Table 2: Department Attributes

Attribute	**Data Type**	**Constraints**
Dept_ID	int	Primary Key
Dept_Head	varchar(20)	
Dept_Name	varchar(15)	

4.3 Staff

Description: Represents hospital employees, including doctors, nurses, and technicians. Stores personal and employment details.

Table 3: Staff Attributes

Attribute	**Data Type**	**Constraints**
Emp_ID	int	Primary Key
Emp_FName	varchar(20)	
Emp_LName	varchar(20)	
Date_Joining	date	
Date_Seperation	date	
Email	varchar(50)	
Address	varchar(50)	
Dept_ID	int	Foreign Key (Department.Dept_ID)
Gender	varchar(10)	

4.4 Doctor

Description: Represents medical doctors, a subset of staff with specific qualifications and specializations.

Table 4: Doctor Attributes

Attribute	Data Type	Constraints
Doctor_ID	int	Primary Key
Qualifications	varchar(20)	
Emp_ID	int	Foreign Key (Staff.Emp_ID)
Specialization	varchar(30)	

4.5 Nurse

Description: Represents nurses, a subset of staff, assigned to specific patients for care.

Table 5: Nurse Attributes

Attribute	Data Type	Constraints
Nurse_ID	int	Primary Key
Patient_ID	int	Foreign Key (Patient.Patient_ID)
Emp_ID	int	Foreign Key (Staff.Emp_ID)

4.6 Emergency_Contact

Description: Stores information about emergency contacts for patients, such as family members or guardians.

Table 6: Emergency_Contact Attributes

Attribute	Data Type	Constraints
Contact_ID	int	Primary Key
Contact_Name	varchar(20)	
Phone	varchar(13)	
Relation	varchar(20)	
Patient_ID	int	Foreign Key (Patient.Patient_ID)

4.7 Payroll

Description: Manages financial details for staff, including salary and bonuses.

Table 7: Payroll Attributes

Attribute	Data Type	Constraints
Account_No	varchar(25)	Primary Key
Salary	decimal(10,2)	
Bonus	decimal(10,2)	
Emp_ID	int	Foreign Key (Staff.Emp_ID)

4.8 Lab_Screening

Description: Records laboratory tests or screenings performed on patients, including technician and doctor details.

Table 8: Lab_Screening Attributes

Attribute	Data Type	Constraints
Lab_ID	int	Primary Key
Patient_ID	int	Foreign Key (Patient.Patient_ID)
Technician_ID	int	Foreign Key (Staff.Emp_ID)
Doctor_ID	int	Foreign Key (Doctor.Doctor_ID)
Test_Cost	decimal(10,2)	
Date	date	

4.9 Insurance

Description: Stores details about patients' insurance policies, including provider and coverage information.

Table 9: Insurance Attributes

Attribute	Data Type	Constraints
Policy_Number	varchar(30)	Primary Key
Patient_ID	int	Foreign Key (Patient.Patient_ID)
Insurer_IRDAI_Code	varchar(15)	
Start_Date	date	
End_Date	date	
Provider_Name	varchar(100)	
Plan_Name	varchar(100)	
Sum_Insured	decimal(12,2)	

4.10 Medicine

Description: Represents medications available in the hospital's pharmacy, including stock and pricing.

Table 10: Medicine Attributes

Attribute	Data Type	Constraints
Medicine_ID	int	Primary Key
M_Name	varchar(20)	
M_Quantity	int	
M_Cost	decimal(10,2)	

4.11 Prescription

Description: Records medications prescribed to patients by doctors, including dosage and date.

Table 11: Prescription Attributes

Attribute	Data Type	Constraints
Prescription_ID	int	Primary Key
Patient_ID	int	Foreign Key (Patient.Patient_ID)
Medicine_ID	int	Foreign Key (Medicine.Medicine_ID)
Date	date	
Dosage	int	
Doctor_ID	int	Foreign Key (Doctor.Doctor_ID)

4.12 Medical_History

Description: Stores patients' past medical records, including allergies and pre-existing conditions.

Table 12: Medical_History Attributes

Attribute	Data Type	Constraints
Record_ID	int	Primary Key
Patient_ID	int	Foreign Key (Patient.Patient_ID)
Allergies	varchar(50)	
Pre_Conditions	varchar(50)	

4.13 Appointment

Description: Manages scheduled appointments between patients and doctors, including date and day of the week.

Table 13: Appointment Attributes

Attribute	Data Type	Constraints
Appt_ID	int	Primary Key
Date	date	
Day_Of_Week	enum(Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday)	
Doctor_ID	int	Foreign Key (Doctor.Doctor
Patient_ID	int	Foreign Key (Patient.Patient

4.14 Room

Description: Represents hospital rooms assigned to patients, including type and associated costs.

Table 14: Room Attributes

Attribute	Data Type	Constraints
Room_ID	int	Primary Key
Room_Type	varchar(50)	
Patient_ID	int	Foreign Key (Patient.Patient_ID)
Doctor_ID	int	Foreign Key (Doctor.Doctor_ID)
Room_Cost	decimal(10,2)	

4.15 Bill

Description: Manages billing details for patients, including costs for rooms, tests, medications, and insurance coverage.

Table 15: Bill Attributes

Attribute	Data Type	Constraints
Bill_ID	int	Primary Key
Date	date	
Room_Cost	decimal(10,2)	
Test_Cost	decimal(10,2)	
Other_Charges	decimal(10,2)	
M_Cost	decimal(10,2)	
Total	decimal(10,2)	
Patient_ID	int	Foreign Key (Patient.Patient_ID)
Remaining_Balance	decimal(10,2)	
Policy_Number	varchar(30)	Foreign Key (Insurance.Policy_Number)

4.16 Doctor_Schedule

Description: Stores doctors' availability schedules, including days and time slots.

Table 16: Doctor_Schedule Attributes

Attribute	**Data Type**	**Constraints**
Schedule_ID	int	Primary Key, Auto-incremen
Doctor_ID	int	Foreign Key (Doctor.Doctor
Day_Of_Week	enum(Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday)	
Start_Time	time	
End_Time	time	
Slot_Duration	int	

5 Relationships

The following table lists the 21 relationships between entities, defined by foreign key constraints. Each relationship specifies the referencing table, the referenced table, the foreign key, and a brief description.

Table 17: Relationships in HospitalDB

Referencing Table	**Referenced Table**	**Foreign Key**	**Description**
Staff	Department	Dept_ID	Associates each staff member with their department.
Doctor	Staff	Emp_ID	Links a doctor to their staff record.
Nurse	Patient	Patient_ID	Indicates the patient a nurse is assigned to.
Nurse	Staff	Emp_ID	Links a nurse to their staff record.
Emergency_Contact	Patient	Patient_ID	Associates an emergency contact with a patient.
Payroll	Staff	Emp_ID	Links payroll records to a staff member.
Lab_Screening	Patient	Patient_ID	Indicates the patient undergoing a lab screening.
Lab_Screening	Staff	Technician_ID	Identifies the technician conducting the screening.
Lab_Screening	Doctor	Doctor_ID	Links the screening to the overseeing doctor.
Insurance	Patient	Patient_ID	Associates an insurance policy with a patient.
Prescription	Patient	Patient_ID	Indicates the patient receiving a prescription.
Prescription	Medicine	Medicine_ID	Links a prescription to the prescribed medicine.
Prescription	Doctor	Doctor_ID	Identifies the doctor issuing the prescription.
Medical_History	Patient	Patient_ID	Associates medical history with a patient.
Appointment	Doctor	Doctor_ID	Links an appointment to the scheduled doctor.
Appointment	Patient	Patient_ID	Indicates the patient for the appointment.
Room	Patient	Patient_ID	Associates a room with the occupying patient.
Room	Doctor	Doctor_ID	Links a room to the responsible doctor.
Bill	Patient	Patient_ID	Associates a bill with the patient being charged.
Bill	Insurance	Policy_Number	Links a bill to the covering insurance policy.
Doctor_Schedule	Doctor	Doctor_ID	Associates a schedule with a doctor.

6 Few Notes and ER model

- The `Condition` field in the `Patient` table was renamed from `Condition_` to avoid reserved keywords.
- The `Policy_Number` in `Bill` is set to `varchar(30)` to match `Insurance.Policy_Number`, resolving a potential inconsistency in the original schema (`varchar(20)`).
- Foreign keys (e.g., `Room.Patient_ID`) may be nullable in practice (e.g., for unoccupied rooms), but nullability is not explicitly defined in the schema.

6.1 ER model

Constraints and Integrity

- Primary keys ensure entity uniqueness.
- Foreign keys maintain relational consistency across entities.
- NOT NULL constraints are used where values are mandatory (e.g., patient name, phone).

6.2 ER diagram

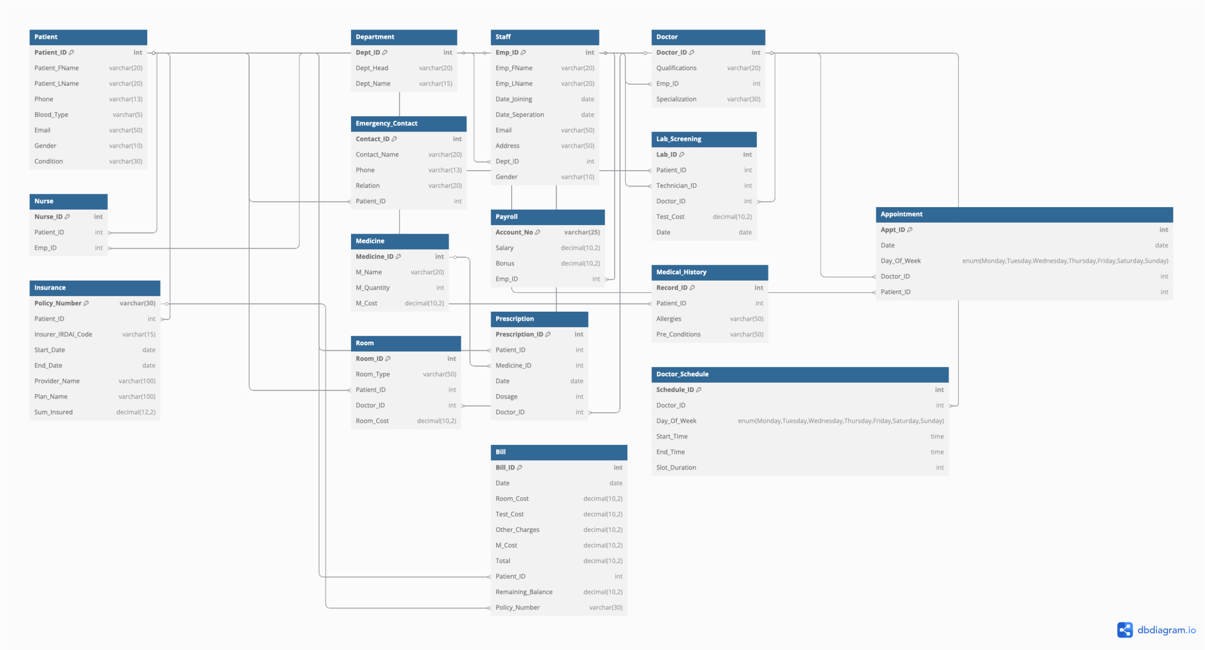


Figure 1: ER diagram

7 Normalization

The following analysis evaluates each table in the `HospitalDB` schema for compliance with First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF), including functional dependencies.

1. Patient

Attributes: `Patient_ID` (PK), `Patient_FName`, `Patient_LName`, `Phone`, `Blood_Type`, `Email`, `Gender`, `Condition`

Functional Dependencies:

- `Patient_ID` → `Patient_FName, Patient_LName, Phone, Blood_Type, Email, Gender, Condition`

Normal Form Analysis:

- **1NF**: Atomic, no repeating groups, unique key
- **2NF**: Single-column primary key `Patient_ID` ensures no partial dependencies.
- **3NF**: No transitive dependencies; all attributes depend directly on `Patient_ID`.

2. Department

Attributes: `Dept_ID` (PK), `Dept_Head`, `Dept_Name`

Functional Dependencies:

- `Dept_ID → Dept_Head, Dept_Name`

Normal Form Analysis:

- **1NF**: Atomic, no repeating groups, unique key
- **2NF**: Single-column primary key `Dept_ID` prevents partial dependencies.
- **3NF**: No transitive dependencies; attributes depend only on `Dept_ID`.

3. Staff

Attributes: `Emp_ID` (PK), `Emp_FName`, `Emp_LName`, `Date_Joining`, `Date_Seperation`, `Email`, `Address`, `Dept_ID`, `Gender`

Functional Dependencies:

- `Emp_ID → Emp_FName, Emp_LName, Date_Joining, Date_Seperation, Email, Address, Dept_ID, Gender`

Normal Form Analysis:

- **1NF**: Atomic, no repeating groups, unique key
- **2NF**: Single-column primary key `Emp_ID` ensures no partial dependencies.
- **3NF**: No transitive dependencies; all attributes depend on `Emp_ID`.

4. Doctor

Attributes: `Doctor_ID` (PK), `Qualifications`, `Emp_ID`, `Specialization`

Functional Dependencies:

- `Doctor_ID → Qualifications, Emp_ID, Specialization`
- `Emp_ID → Doctor_ID`

Normal Form Analysis:

- **1NF**: Atomic, no repeating groups, unique key
- **2NF**: Single-column primary key `Doctor_ID` prevents partial dependencies.
- **3NF**: No transitive dependencies; `Emp_ID` is a foreign key, not a transitive issue.

5. Nurse

Attributes: `Nurse_ID` (PK), `Patient_ID`, `Emp_ID`

Functional Dependencies:

- `Nurse_ID → Patient_ID, Emp_ID`
- `Emp_ID → Nurse_ID`

Normal Form Analysis:

- **1NF**: Atomic, no repeating groups, unique key
- **2NF**: Single-column primary key `Nurse_ID` ensures no partial dependencies.
- **3NF**: No transitive dependencies; attributes depend on `Nurse_ID`.

6. Emergency_Contact

Attributes: Contact_ID (PK), Contact_Name, Phone, Relation, Patient_ID
Functional Dependencies:

- Contact_ID → Contact_Name, Phone, Relation, Patient_ID

Normal Form Analysis:

- **1NF**: Atomic, no repeating groups, unique key
- **2NF**: Single-column primary key Contact_ID prevents partial dependencies.
- **3NF**: No transitive dependencies; all attributes depend on Contact_ID.

7. Payroll

Attributes: Account_No (PK), Salary, Bonus, Emp_ID
Functional Dependencies:

- Account_No → Salary, Bonus, Emp_ID

Normal Form Analysis:

- **1NF**: Atomic, no repeating groups, unique key
- **2NF**: Single-column primary key Account_No ensures no partial dependencies.
- **3NF**: No transitive dependencies; attributes depend on Account_No.

8. Lab_Screening

Attributes: Lab_ID (PK), Patient_ID, Technician_ID, Doctor_ID, Test_Cost, Date
Functional Dependencies:

- Lab_ID → Patient_ID, Technician_ID, Doctor_ID, Test_Cost, Date

Normal Form Analysis:

- **1NF**: Atomic, no repeating groups, unique key
- **2NF**: Single-column primary key Lab_ID prevents partial dependencies.
- **3NF**: No transitive dependencies; all attributes depend on Lab_ID.

9. Insurance

Attributes: Policy_Number (PK), Patient_ID, Insurer_IRDAI_Code, Start_Date, End_Date, Provider_Name, Plan_Name, Sum_Insured
Functional Dependencies:

- Policy_Number → Patient_ID, Insurer_IRDAI_Code, Start_Date, End_Date, Provider_Name, Plan_Name, Sum_Insured

Normal Form Analysis:

- **1NF**: Atomic, no repeating groups, unique key
- **2NF**: Single-column primary key Policy_Number ensures no partial dependencies.
- **3NF**: No transitive dependencies, assuming Provider_Name is tied to Policy_Number.

10. Medicine

Attributes: `Medicine_ID` (PK), `M_Name`, `M_Quantity`, `M_Cost`
Functional Dependencies:

- `Medicine_ID → M_Name, M_Quantity, M_Cost`

Normal Form Analysis:

- **1NF**: Atomic, no repeating groups, unique key
- **2NF**: Single-column primary key `Medicine_ID` prevents partial dependencies.
- **3NF**: No transitive dependencies; all attributes depend on `Medicine_ID`.

11. Prescription

Attributes: `Prescription_ID` (PK), `Patient_ID`, `Medicine_ID`, `Date`, `Dosage`, `Doctor_ID`
Functional Dependencies:

- `Prescription_ID → Patient_ID, Medicine_ID, Date, Dosage, Doctor_ID`

Normal Form Analysis:

- **1NF**: Atomic, no repeating groups, unique key
- **2NF**: Single-column primary key `Prescription_ID` ensures no partial dependencies.
- **3NF**: No transitive dependencies; all attributes depend on `Prescription_ID`.

12. Medical_History

Attributes: `Record_ID` (PK), `Patient_ID`, `Allergies`, `Pre_Conditions`
Functional Dependencies:

- `Record_ID → Patient_ID, Allergies, Pre_Conditions`

Normal Form Analysis:

- **1NF**: Atomic, no repeating groups, unique key
- **2NF**: Single-column primary key `Record_ID` prevents partial dependencies.
- **3NF**: No transitive dependencies; all attributes depend on `Record_ID`.

13. Appointment

Attributes: `Appt_ID` (PK), `Date`, `Day_Of_Week`, `Doctor_ID`, `Patient_ID`
Functional Dependencies:

- `Appt_ID → Date, Day_Of_Week, Doctor_ID, Patient_ID`
- `Date, Doctor_ID → Day_Of_Week`

Normal Form Analysis:

- **1NF**: Atomic, no repeating groups, unique key
- **2NF**: Single-column primary key `Appt_ID` ensures no partial dependencies.
- **3NF**: Transitive dependency `Date → Day_Of_Week` violates 3NF; `Day_Of_Week` is derivable.

14. Room

Attributes: `Room_ID` (PK), `Room_Type`, `Patient_ID`, `Doctor_ID`, `Room_Cost`

Functional Dependencies:

- `Room_ID → Room_Type, Patient_ID, Doctor_ID, Room_Cost`
- `Room_Type → Room_Cost`

Normal Form Analysis:

- **1NF**: Atomic, no repeating groups, unique key
- **2NF**: Single-column primary key `Room_ID` prevents partial dependencies.
- **3NF**: Transitive dependency `Room_Type → Room_Cost` violates 3NF; cost depends on type.

15. Bill

Attributes: `Bill_ID` (PK), `Date`, `Room_Cost`, `Test_Cost`, `Other_Charges`, `M_Cost`, `Total`, `Patient_ID`, `Remaining_Balance`, `Policy_Number`

Functional Dependencies:

- `Bill_ID → Date, Room_Cost, Test_Cost, Other_Charges, M_Cost, Total, Patient_ID, Remaining_Balance, Policy_Number`
- `Room_Cost, Test_Cost, Other_Charges, M_Cost → Total`

Normal Form Analysis:

- **1NF**: Atomic, no repeating groups, unique key
- **2NF**: Single-column primary key `Bill_ID` ensures no partial dependencies.
- **3NF**: Transitive dependency `Room_Cost, Test_Cost, Other_Charges, M_Cost → Total` violates 3NF; `Total` is derivable.

16. Doctor_Schedule

Attributes: `Schedule_ID` (PK, increment), `Doctor_ID`, `Day_Of_Week`, `Start_Time`, `End_Time`, `Slot_Duration`

Functional Dependencies:

- `Schedule_ID → Doctor_ID, Day_Of_Week, Start_Time, End_Time, Slot_Duration`

Normal Form Analysis:

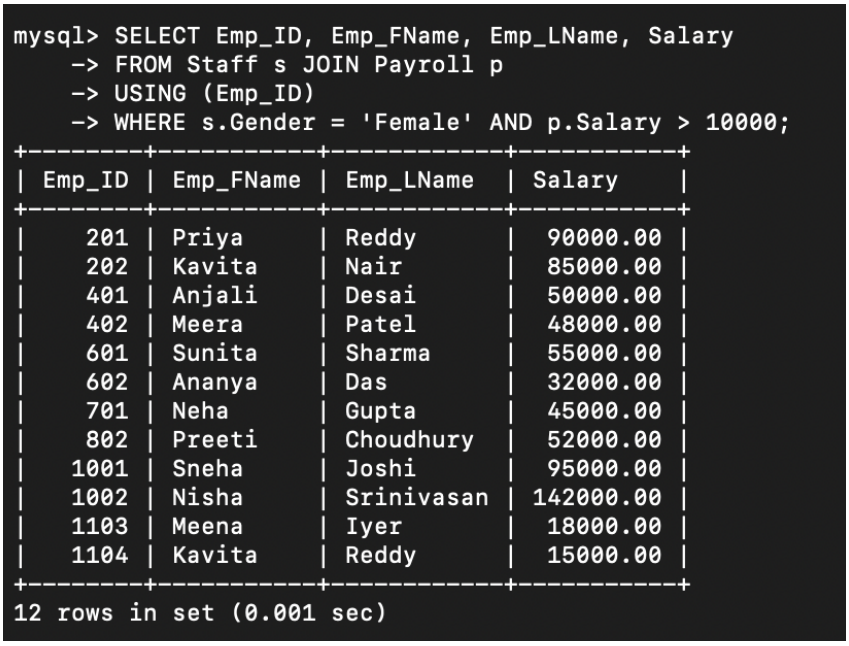
- **1NF**: Atomic, no repeating groups, unique key
- **2NF**: Single-column primary key `Schedule_ID` prevents partial dependencies.
- **3NF**: No transitive dependencies; all attributes depend on `Schedule_ID`.

Summary

All tables are in 1NF and 2NF due to atomic attributes, no repeating groups, and single-column primary keys. Most tables are in 3NF, except `Appointment` (`Date → Day_Of_Week`), `Room` (`Room_Type → Room_Cost`), and `Bill` (`Room_Cost, Test_Cost, Other_Charges, M_Cost → Total`), which have transitive dependencies.

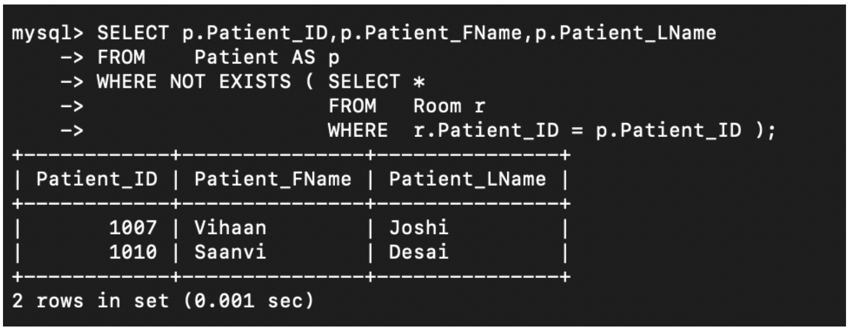
8 Implementation and Results

The database was implemented in SQL using standard DDL (Data Definition Language) statements. We began the design with a *single, all-purpose "mega table"* that tried to hold *every* facts about patients, staff, rooms, billing, and clinical events. Then we kept splitting the database removing redundancy and finally reaching 3NF for most of the tables.



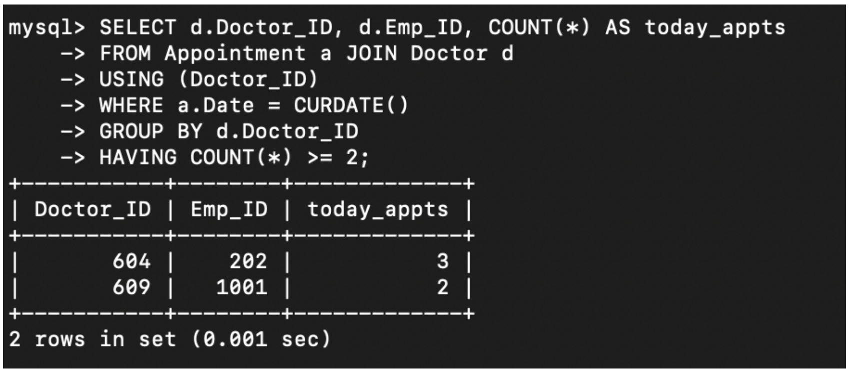
```
mysql> SELECT Emp_ID, Emp_FName, Emp_LName, Salary
    -> FROM Staff s JOIN Payroll p
    -> USING (Emp_ID)
    -> WHERE s.Gender = 'Female' AND p.Salary > 10000;
+--------+-----------+------------+-----------+
| Emp_ID | Emp_FName | Emp_LName  | Salary    |
+--------+-----------+------------+-----------+
|    201 | Priya     | Reddy      |  90000.00 |
|    202 | Kavita    | Nair       |  85000.00 |
|    401 | Anjali    | Desai      |  50000.00 |
|    402 | Meera     | Patel      |  48000.00 |
|    601 | Sunita    | Sharma     |  55000.00 |
|    602 | Ananya    | Das        |  32000.00 |
|    701 | Neha      | Gupta      |  45000.00 |
|    802 | Preeti    | Choudhury  |  52000.00 |
|   1001 | Sneha     | Joshi      |  95000.00 |
|   1002 | Nisha     | Srinivasan | 142000.00 |
|   1103 | Meena     | Iyer       |  18000.00 |
|   1104 | Kavita    | Reddy      |  15000.00 |
+--------+-----------+------------+-----------+
12 rows in set (0.001 sec)
```

Figure 2: Female employees whose monthly salary exceeds Rs. 10000



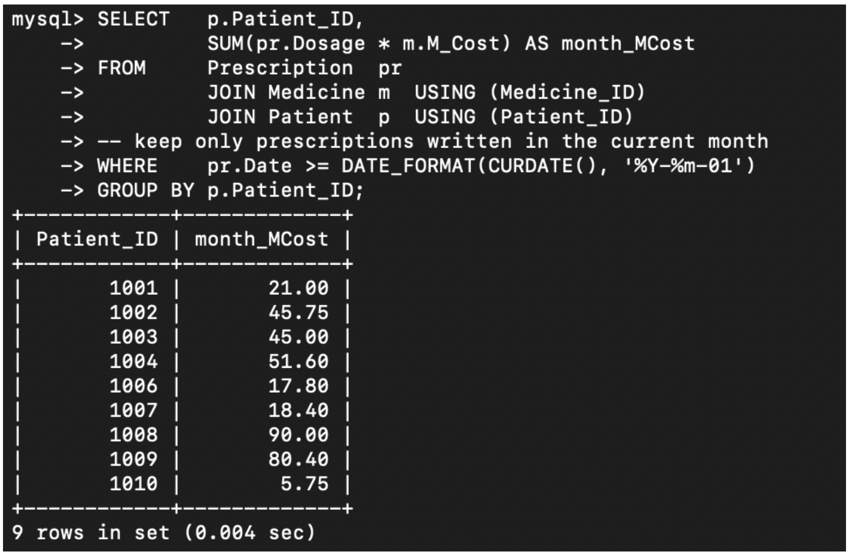
```
mysql> SELECT p.Patient_ID,p.Patient_FName,p.Patient_LName
    -> FROM    Patient AS p
    -> WHERE NOT EXISTS ( SELECT *
    ->                    FROM   Room r
    ->                    WHERE  r.Patient_ID = p.Patient_ID );
+------------+---------------+---------------+
| Patient_ID | Patient_FName | Patient_LName |
+------------+---------------+---------------+
|       1007 | Vihaan        | Joshi         |
|       1010 | Saanvi        | Desai         |
+------------+---------------+---------------+
2 rows in set (0.001 sec)
```

Figure 3: Patient not currently admitted in any rooms



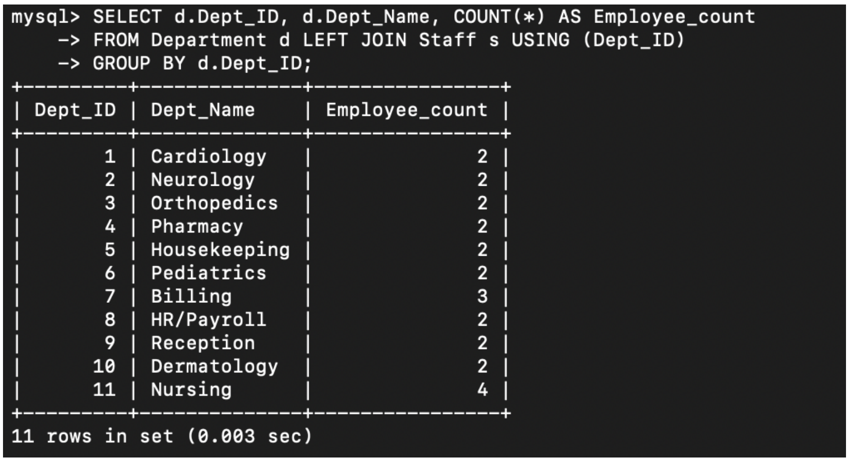
```
mysql> SELECT d.Doctor_ID, d.Emp_ID, COUNT(*) AS today_appts
    -> FROM Appointment a JOIN Doctor d
    -> USING (Doctor_ID)
    -> WHERE a.Date = CURDATE()
    -> GROUP BY d.Doctor_ID
    -> HAVING COUNT(*) >= 2;
+-----------+--------+-------------+
| Doctor_ID | Emp_ID | today_appts |
+-----------+--------+-------------+
|       604 |    202 |           3 |
|       609 |   1001 |           2 |
+-----------+--------+-------------+
2 rows in set (0.001 sec)
```

Figure 4: Doctors whose today’s appointment is greater than equal to 2



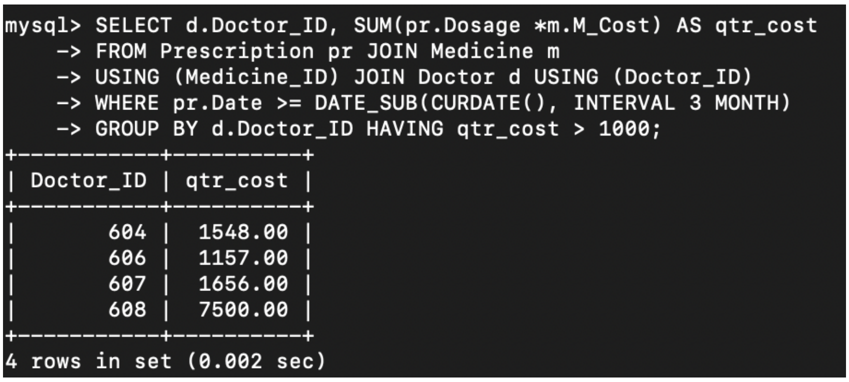
```
mysql> SELECT   p.Patient_ID,
    ->          SUM(pr.Dosage * m.M_Cost) AS month_MCost
    -> FROM     Prescription  pr
    ->          JOIN Medicine m  USING (Medicine_ID)
    ->          JOIN Patient  p  USING (Patient_ID)
    -> -- keep only prescriptions written in the current month
    -> WHERE    pr.Date >= DATE_FORMAT(CURDATE(), '%Y-%m-01')
    -> GROUP BY p.Patient_ID;
+------------+-------------+
| Patient_ID | month_MCost |
+------------+-------------+
|       1001 |       21.00 |
|       1002 |       45.75 |
|       1003 |       45.00 |
|       1004 |       51.60 |
|       1006 |       17.80 |
|       1007 |       18.40 |
|       1008 |       90.00 |
|       1009 |       80.40 |
|       1010 |        5.75 |
+------------+-------------+
9 rows in set (0.004 sec)
```

Figure 5: All patient’s current month medicinal cost



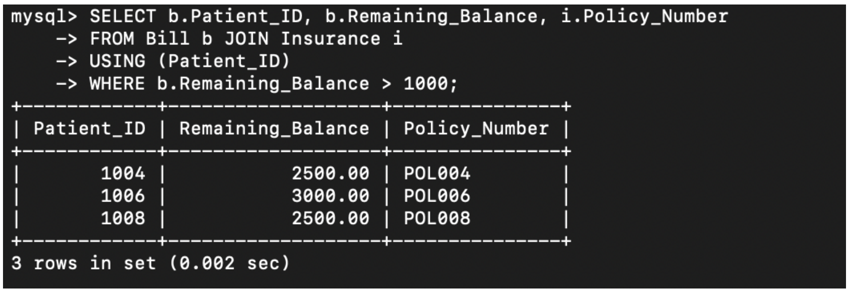
```
mysql> SELECT d.Dept_ID, d.Dept_Name, COUNT(*) AS Employee_count
    -> FROM Department d LEFT JOIN Staff s USING (Dept_ID)
    -> GROUP BY d.Dept_ID;
+---------+-------------+----------------+
| Dept_ID | Dept_Name   | Employee_count |
+---------+-------------+----------------+
|       1 | Cardiology  |              2 |
|       2 | Neurology   |              2 |
|       3 | Orthopedics |              2 |
|       4 | Pharmacy    |              2 |
|       5 | Housekeeping|              2 |
|       6 | Pediatrics  |              2 |
|       7 | Billing     |              3 |
|       8 | HR/Payroll  |              2 |
|       9 | Reception   |              2 |
|      10 | Dermatology |              2 |
|      11 | Nursing     |              4 |
+---------+-------------+----------------+
11 rows in set (0.003 sec)
```

Figure 6: Employee count in each department



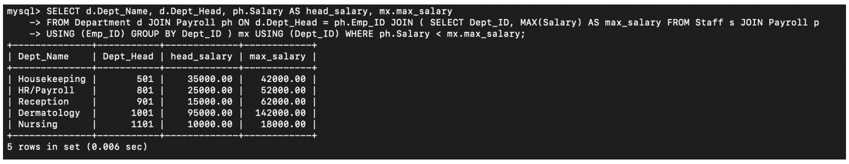
```
mysql> SELECT d.Doctor_ID, SUM(pr.Dosage *m.M_Cost) AS qtr_cost
    -> FROM Prescription pr JOIN Medicine m
    -> USING (Medicine_ID) JOIN Doctor d USING (Doctor_ID)
    -> WHERE pr.Date >= DATE_SUB(CURDATE(), INTERVAL 3 MONTH)
    -> GROUP BY d.Doctor_ID HAVING qtr_cost > 1000;
+-----------+----------+
| Doctor_ID | qtr_cost |
+-----------+----------+
|       604 |  1548.00 |
|       606 |  1157.00 |
|       607 |  1656.00 |
|       608 |  7500.00 |
+-----------+----------+
4 rows in set (0.002 sec)
```

Figure 7: Doctors who have given prescription worth more than Rs.1000 in last three months



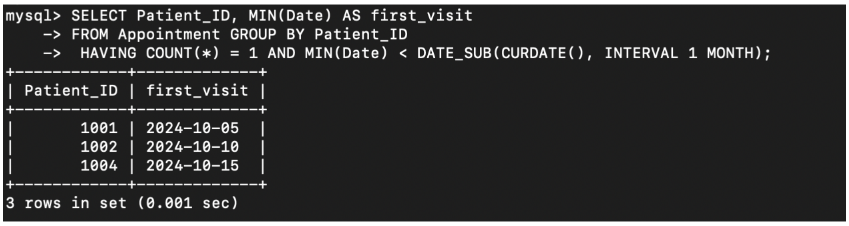
```
mysql> SELECT b.Patient_ID, b.Remaining_Balance, i.Policy_Number
    -> FROM Bill b JOIN Insurance i
    -> USING (Patient_ID)
    -> WHERE b.Remaining_Balance > 1000;
+------------+-------------------+---------------+
| Patient_ID | Remaining_Balance | Policy_Number |
+------------+-------------------+---------------+
|       1004 |           2500.00 | POL004        |
|       1006 |           3000.00 | POL006        |
|       1008 |           2500.00 | POL008        |
+------------+-------------------+---------------+
3 rows in set (0.002 sec)
```

Figure 8: Patient whose remaining balance is more than 1000 holding an insurance



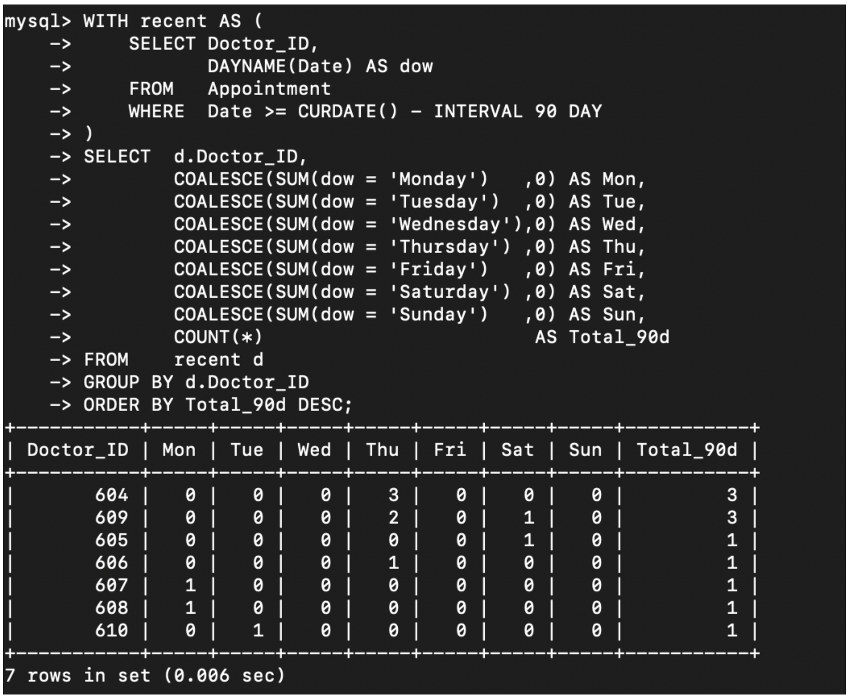
```
mysql> SELECT d.Dept_Name, d.Dept_Head, ph.Salary AS head_salary, mx.max_salary
    -> FROM Department d JOIN Payroll ph ON d.Dept_Head = ph.Emp_ID JOIN ( SELECT Dept_ID, MAX(Salary) AS max_salary FROM Staff s JOIN Payroll p
    -> USING (Emp_ID) GROUP BY Dept_ID ) mx USING (Dept_ID) WHERE ph.Salary < mx.max_salary;
+--------------+-----------+-------------+------------+
| Dept_Name    | Dept_Head | head_salary | max_salary |
+--------------+-----------+-------------+------------+
| Housekeeping |       501 |    35000.00 |   42000.00 |
| HR/Payroll   |       801 |    25000.00 |   52000.00 |
| Reception    |       901 |    15000.00 |   62000.00 |
| Dermatology  |      1001 |    95000.00 |  142000.00 |
| Nursing      |      1101 |    10000.00 |   18000.00 |
+--------------+-----------+-------------+------------+
5 rows in set (0.006 sec)
```

Figure 9: Departments in which it’s heads salary is not highest.



```
mysql> SELECT Patient_ID, MIN(Date) AS first_visit
    -> FROM Appointment GROUP BY Patient_ID
    ->  HAVING COUNT(*) = 1 AND MIN(Date) < DATE_SUB(CURDATE(), INTERVAL 1 MONTH);
+------------+-------------+
| Patient_ID | first_visit |
+------------+-------------+
|       1001 | 2024-10-05  |
|       1002 | 2024-10-10  |
|       1004 | 2024-10-15  |
+------------+-------------+
3 rows in set (0.001 sec)
```

Figure 10: Patient whose first visit was more than 1 months back



```
mysql> WITH recent AS (
    ->     SELECT Doctor_ID,
    ->            DAYNAME(Date) AS dow
    ->     FROM   Appointment
    ->     WHERE  Date >= CURDATE() - INTERVAL 90 DAY
    -> )
    -> SELECT  d.Doctor_ID,
    ->         COALESCE(SUM(dow = 'Monday')   ,0) AS Mon,
    ->         COALESCE(SUM(dow = 'Tuesday')  ,0) AS Tue,
    ->         COALESCE(SUM(dow = 'Wednesday'),0) AS Wed,
    ->         COALESCE(SUM(dow = 'Thursday') ,0) AS Thu,
    ->         COALESCE(SUM(dow = 'Friday')   ,0) AS Fri,
    ->         COALESCE(SUM(dow = 'Saturday') ,0) AS Sat,
    ->         COALESCE(SUM(dow = 'Sunday')   ,0) AS Sun,
    ->         COUNT(*)                          AS Total_90d
    -> FROM    recent d
    -> GROUP BY d.Doctor_ID
    -> ORDER BY Total_90d DESC;
+-----------+-----+-----+-----+-----+-----+-----+-----+-----------+
| Doctor_ID | Mon | Tue | Wed | Thu | Fri | Sat | Sun | Total_90d |
+-----------+-----+-----+-----+-----+-----+-----+-----+-----------+
|       604 |   0 |   0 |   0 |   3 |   0 |   0 |   0 |         3 |
|       609 |   0 |   0 |   0 |   2 |   0 |   1 |   0 |         3 |
|       605 |   0 |   0 |   0 |   0 |   0 |   1 |   0 |         1 |
|       606 |   0 |   0 |   0 |   1 |   0 |   0 |   0 |         1 |
|       607 |   1 |   0 |   0 |   0 |   0 |   0 |   0 |         1 |
|       608 |   1 |   0 |   0 |   0 |   0 |   0 |   0 |         1 |
|       610 |   0 |   1 |   0 |   0 |   0 |   0 |   0 |         1 |
+-----------+-----+-----+-----+-----+-----+-----+-----+-----------+
7 rows in set (0.006 sec)
```

Figure 11: For each doctor: how many appointments per weekday in the last 90 days, pivoted across columns

9 Discussion and Limitations

9.1 Strengths

The `HospitalDB` database schema exhibits several strengths that make it suitable for managing hospital operations, particularly for small to medium-sized facilities. These strengths are outlined below:

- **Comprehensive Entity Coverage**: The schema includes 16 entities (e.g., `Patient`, `Staff`, `Bill`) that cover critical hospital functions, such as patient care, staff management, billing, and scheduling. This holistic design centralizes data management, reducing the need for multiple systems.
- **Well-Defined Relationships**: 21 foreign key relationships (e.g., `Doctor.Emp_ID` → `Staff.Emp_ID`) ensure referential integrity and model hospital workflows accurately, enabling precise data retrieval and reporting.
- **Support for Key Hospital Processes**: The schema supports patient care (e.g., `Prescription`, `Medical_History`), billing (e.g., `Bill`, `Insurance`), staff management (e.g., `Payroll`), and scheduling (e.g., `Appointment`), streamlining operations.
- **Scalability for Core Operations**: Primary keys and auto-incrementing fields (e.g., `Schedule_ID`) support growth in data volume, making the schema suitable for small to medium hospitals.
- **Database-Agnostic Design**: While specified for MySQL, the schema's standard SQL structure allows adaptation to other databases (e.g., PostgreSQL), reducing vendor lock-in.

9.2 Limitations

Despite its strengths, the `HospitalDB` schema has limitations that may impact its performance, scalability, or suitability for complex hospital environments. These limitations are detailed below:

- **Lack of Indexes for Performance**: Beyond primary keys, no indexes are defined for frequently queried fields (e.g., `Patient_ID`, `Date`), potentially slowing queries in high-traffic environments.
- **Limited Support for Complex Scheduling**: `Doctor_Schedule` lacks fields for exceptions (e.g., holidays) or appointment status, limiting dynamic scheduling capabilities.
- **Incomplete Staff Role Differentiation**: The schema does not model roles like pharmacists or administrators beyond `Doctor` and `Nurse`, restricting role-specific workflows.
- **No Audit or History Tracking**: Absence of audit tables or fields (e.g., `Created_At`, `Updated_By`) hinders compliance with regulations like HIPAA and complicates error tracking.
- **Simplified Billing Model**: `Bill` aggregates costs without linking to source tables (e.g., `Room`, `Prescription`), risking redundancy or inconsistencies.
- **Scalability for Large Hospitals**: The schema lacks partitioning or concurrency optimizations, potentially causing performance issues in large hospitals with millions of records.
- **Limited Support for Advanced Features**: The schema does not support multi-hospital setups, patient transfers, or external system integrations, requiring modifications for enterprise use.

The `HospitalDB` schema is a robust and comprehensive solution for small to medium-sized hospitals, offering well-defined relationships, flexible data types, and support for core operations like patient care, billing, and scheduling. Its database-agnostic design and scalability for moderate data volumes enhance its applicability. However, limitations such as missing indexes, unclear nullability, lack of audit tracking, and simplified models for billing and scheduling may pose challenges in high-traffic or complex environments. Additional constraints, role-specific tables, and advanced features (e.g., multi-hospital support) would be needed to address these gaps and ensure scalability and compliance in larger or enterprise-level hospital systems.

10 Contributions

Rishikesh Sahil - 25% Kawaljeet Singh - 25% Nikhil Jain - 25% Abhishek Kumar - 25%